AgEcon SEARCH
RESEARCH IN AGRICULTURAL & APPLIED ECONOMICS

The World's Largest Open Access Agricultural & Applied Economics Digital Library

This document is discoverable and free to researchers across the globe due to the work of AgEcon Search.

Help ensure our sustainability.

Give to AgEcon Search

AgEcon Search
http://ageconsearch.umn.edu
aesearch@umn.edu

*Papers downloaded from **AgEcon Search** may be used for non-commercial purposes and personal study only. No other use, including posting to another Internet site, is permitted without permission from the copyright owner (not AgEcon Search), or as allowed under the provisions of Fair Use, U.S. Copyright Act, Title 17 U.S.C.*

No endorsement of AgEcon Search or its fundraising activities by the author(s) of the following work or their employer(s) is intended or implied.

The Stata Journal

Editor
H. Joseph Newton
Department of Statistics
Texas A&M University
College Station, Texas 77843
979-845-8817; fax 979-845-6077
jnewton@stata-journal.com

Editor
Nicholas J. Cox
Department of Geography
Durham University
South Road
Durham DH1 3LE UK
n.j.cox@stata-journal.com

Associate Editors

Christopher F. Baum
Boston College

Nathaniel Beck
New York University

Rino Bellocco
Karolinska Institutet, Sweden, and
University of Milano-Bicocca, Italy

Maarten L. Buis
Tübingen University, Germany

A. Colin Cameron
University of California–Davis

Mario A. Cleves
Univ. of Arkansas for Medical Sciences

William D. Dupont
Vanderbilt University

David Epstein
Columbia University

Allan Gregory
Queen's University

James Hardin
University of South Carolina

Ben Jann
University of Bern, Switzerland

Stephen Jenkins
London School of Economics and
Political Science

Ulrich Kohler
WZB, Berlin

Frauke Kreuter
University of Maryland–College Park

Peter A. Lachenbruch
Oregon State University

Jens Lauritsen
Odense University Hospital

Stanley Lemeshow
Ohio State University

J. Scott Long
Indiana University

Roger Newson
Imperial College, London

Austin Nichols
Urban Institute, Washington DC

Marcello Pagano
Harvard School of Public Health

Sophia Rabe-Hesketh
University of California–Berkeley

J. Patrick Royston
MRC Clinical Trials Unit, London

Philip Ryan
University of Adelaide

Mark E. Schaffer
Heriot-Watt University, Edinburgh

Jeroen Weesie
Utrecht University

Nicholas J. G. Winter
University of Virginia

Jeffrey Wooldridge
Michigan State University

Stata Press Editorial Manager Lisa Gilmore
Stata Press Copy Editors Deirdre Skaggs

The *Stata Journal* publishes reviewed papers together with shorter notes or comments, regular columns, book reviews, and other material of interest to Stata users. Examples of the types of papers include 1) expository papers that link the use of Stata commands or programs to associated principles, such as those that will serve as tutorials for users first encountering a new field of statistics or a major new technique; 2) papers that go "beyond the Stata manual" in explaining key features or uses of Stata that are of interest to intermediate or advanced users of Stata; 3) papers that discuss new commands or Stata programs of interest either to a wide spectrum of users (e.g., in data management or graphics) or to some large segment of Stata users (e.g., in survey statistics, survival analysis, panel analysis, or limited dependent variable modeling); 4) papers analyzing the statistical properties of new or existing estimators and tests in Stata; 5) papers that could be of interest or usefulness to researchers, especially in fields that are of practical importance but are not often included in texts or other journals, such as the use of Stata in managing datasets, especially large datasets, with advice from hard-won experience; and 6) papers of interest to those who teach, including Stata with topics such as extended examples of techniques and interpretation of results, simulations of statistical concepts, and overviews of subject areas.

For more information on the *Stata Journal*, including information for authors, see the webpage

http://www.stata-journal.com

The *Stata Journal* is indexed and abstracted in the following:

- CompuMath Citation Index®
- Current Contents/Social and Behavioral Sciences®
- RePEc: Research Papers in Economics
- Science Citation Index Expanded (also known as SciSearch®)
- Scopus™
- Social Sciences Citation Index®

Copyright Statement: The *Stata Journal* and the contents of the supporting files (programs, datasets, and help files) are copyright © by StataCorp LP. The contents of the supporting files (programs, datasets, and help files) may be copied or reproduced by any means whatsoever, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the *Stata Journal*.

The articles appearing in the *Stata Journal* may be copied or reproduced as printed copies, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the *Stata Journal*.

Written permission must be obtained from StataCorp if you wish to make electronic copies of the insertions. This precludes placing electronic copies of the *Stata Journal*, in whole or in part, on publicly accessible websites, fileservers, or other locations where the copy may be accessed by anyone other than the subscriber.

Users of any of the software, ideas, data, or other materials published in the *Stata Journal* or the supporting files understand that such use is made without warranty of any kind, by either the *Stata Journal*, the author, or StataCorp. In particular, there is no warranty of fitness of purpose or merchantability, nor for special, incidental, or consequential damages such as loss of profits. The purpose of the *Stata Journal* is to promote free communication among Stata users.

The *Stata Journal*, electronic version (ISSN 1536-8734) is a publication of Stata Press. Stata, Mata, NetCourse, and Stata Press are registered trademarks of StataCorp LP.

The Stata Journal (2012)
12, Number 1, pp. 94–129

Stata graph library for network analysis

Hirotaka Miura
Department of Mathematics
Columbia University
New York, NY
hm2528@columbia.edu
hirotaka.miura@gmail.com

Abstract. Network analysis is a multidisciplinary research method that is quickly becoming a popular and exciting field. Though some statistical programs possess sophisticated packages for analyzing networks, similar capabilities have yet to be made available in Stata. In an effort to motivate the use of Stata for network analysis, I designed in Mata the Stata graph library (SGL), which consists of algorithms that construct matrix representations of networks, compute centrality measures, calculate clustering coefficients, and solve maximum-flow problems. The SGL is designed for both directed and undirected one-mode networks containing edges that are either unweighted or weighted with positive values. Performance tests conducted between C++ and Stata graph library implementations indicate gross inefficiencies in current SGL routines, making the SGL impractical for large networks. The obstacles are, however, welcome challenges in the effort to spread the use of Stata for analyzing networks. Future developments will focus toward addressing computational time complexities and integrating additional capabilities into the SGL.

Keywords: st0248, netsis, netsummarize, centrality, clustering, network analysis

1 Introduction

1.1 What is network analysis?

Network analysis is an application of network theory, a subfield of graph theory, that is concerned with analyzing relational data. Some questions network analysis addresses are how important or central individual actors are in a given network and how concentrated the network is. Example uses of network analysis include the following:

- Determining the importance of a webpage using Google's PageRank.
- Examining communication networks in intelligence and computer security.
- Solving transportation problems that involve flow of traffic or commodities.
- Addressing the too-connected-to-fail problem in financial networks.
- Analyzing social relationships between individuals in social network analysis.

© 2012 StataCorp LP

1.2 Modeling relational data

A graph model representing a network $G = (V, E)$ consists of a set of vertices V and a set of edges E. The size of set V, the number of vertices in network G, is denoted $|V|$; similarly, the size of set E, the number of edges in network G, is denoted $|E|$. An edge is defined as a link between two vertices i and j, not necessarily distinct, that has vertex i on one end and vertex j on the other. An edge may be directed or undirected. A network may be weighted, as when two edges are weighted with differing edge values, or it may be unweighted, as when all edges have an edge value of one.

There are special types of vertices and edges that standard graph algorithms cannot handle or for which accommodating routines simply do not exist. Thus the following types of vertices and edges are excluded from analysis:

- Isolated vertex—a vertex that is not attached to any edges.
- Parallel edges—two or more edges that connect the same pair of vertices.
- Self-loop—an edge connecting vertex i to itself.
- Zero- or negative-weighted edge.

A variety of methods exist for capturing relational data, with the adjacency matrix and adjacency list forms being some of the more widely used storage types. In Stata, however, as will be demonstrated in a later section, capturing relational data in a coordinate list or edge list is more advantageous because it allows the user to use Stata's built-in capabilities, such as the ability to restrict the scope of the analysis by specifying `if` *expression* and `in` *range* qualifiers.

1.3 Edge list

An edge list for an undirected, unweighted network is an $|E| \times 2$ matrix where each row represents an edge between vertices i and j. A directed, unweighted network is defined similarly with an $|E| \times 2$ matrix capturing information on directed edges from source vertex i to target vertex j. A weighted network can be represented by adding a third column containing edge weights.

$$\begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 4 & 2 \end{bmatrix}$$

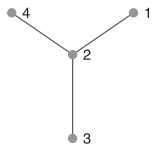


Figure 1. Example of an edge list and its corresponding graph (undirected and unweighted); drawn using `netplot` (Corten 2011)

Substantial modifications may be needed to arrive at an edge list from an initial dataset. The task of modifying initial data applies to datasets in both long and wide formats, as well as data in matrix form.

2 Matrix representation

2.1 Adjacency matrix

Adjacency matrix **A** for unweighted networks is defined as a $|V|\times|V|$ matrix with entries A_{ij} equal to one if an edge connects vertices i and j, and equal to zero otherwise. A_{ii} entries are set to zero, and matrix **A** is symmetric for undirected networks. For directed networks, rows of matrix **A** represent outgoing edges and columns represent incoming edges.[1] For weighted networks, entries A_{ij} are equal to the weight of the edge connecting vertices i and j.

2.2 Distance matrix

Distance matrix **D** is defined as a $|V| \times |V|$ matrix with each entry D_{ij} equal to the length of the shortest path between vertices i and j. A path is defined as a way to reach vertex j from vertex i using a combination of edges that do not go through a particular vertex more than once. If no such path exists between vertices i and j, then D_{ij} is set to missing, signifying what is sometimes called an infinite path. D_{ii} is set to zero. Matrix **D** is symmetric for undirected networks.

2.3 Path matrix

Path matrix **P** is defined as a $|V|\times|V|$ matrix with P_{ij} entries being equal to the number of shortest paths between vertices i and j. If no paths exist between vertices i and j, P_{ij} is set to zero. P_{ii} is set to one. Matrix **P** is symmetric for undirected networks.

3 Centrality measures

3.1 Degree centrality

Degree centrality measures the importance of a vertex by the number of connections the vertex has if the network is unweighted (Freeman 1977), and by the aggregate of the weights of edges connected to the vertex if the network is weighted (Barrat et al. 2004). For an undirected network, degree centrality for vertex i is defined as

$$\frac{1}{|V|-1}\sum_{j\neq i} A_{ij} \tag{1}$$

1. The convention of denoting X_{ij} entries as an edge from i to j is adopted for all matrices.

where the leading divisor is adjusted for excluding the $j = i$ term. Directed networks may entail vertices having a different number of incoming and outgoing edges, and thus we have out-degree and in-degree centrality. Out-degree centrality for vertex i is defined equivalently to (1). For in-degree centrality, we simply transpose the adjacency matrix:

$$\frac{1}{|V| - 1} \sum_{j \neq i} A'_{ij}$$

3.2 Closeness centrality

Closeness centrality provides higher centrality scores to vertices that are situated closer to members of their component (the set of reachable vertices) by taking the inverse of the average shortest paths as a measure of proximity (Freeman 1977). That is, the closeness centrality for vertex i is defined as

$$\frac{(|V| - 1)}{\sum_{j \neq i} D_{ij}} \tag{2}$$

which reflects how vertices with smaller average shortest path lengths receive higher centrality scores than those that are situated farther away from members of their component.

An immediate concern in computing (2) is how to deal with infinite distances to unreachable vertices. A common workaround is to average over only the vertices that are reachable. However, caution must be exercised because distances between vertices tend to be shorter in smaller components, possibly resulting in vertices in such components receiving higher closeness centrality scores than vertices in larger components, going against the notion that vertices in small components are less central in the network (Newman 2010, sec. 7.6).

3.3 Betweenness centrality

Betweenness centrality bestows larger centrality scores on vertices that lie on a larger proportion of shortest paths linking pairs of other vertices. Let P_{ij} denote the number of shortest paths from vertex i to j, as defined above. Let $P_{ij}(k)$ denote the number of shortest paths from vertex i to j containing vertex k. Then following Anthonisse (1971) and Freeman (1977), the betweenness centrality measure for vertex k is defined as

$$\sum_{i \neq j \neq k \neq i} \frac{P_{ij}(k)}{P_{ij}} \tag{3}$$

To normalize (3), divide by $(|V| - 1)(|V| - 2)$, the maximum number of paths that a given vertex could lie on between pairs of other vertices.[2]

2. Actual implementation is completed for undirected networks by calculating both $P_{ij}(k)$ and $P_{ji}(k)$, and thus the numerator does not need to be multiplied by two.

3.4 Eigenvector centrality

Eigenvector centrality can provide an indication of how important a vertex is by having the property of being large if a vertex has many neighbors, important neighbors, or both. The measure first proposed by Bonacich (1972) defines the centrality of vertex i, x_i, as the sum of the centrality of its neighbors multiplied by a constant. That is, for an undirected network with adjacency matrix $\mathbf{A}$,

$$x_i = \lambda^{-1} \sum_j A_{ij} x_j \tag{4}$$

which can be rewritten as

$$\lambda \mathbf{x} = \mathbf{A}\mathbf{x} \tag{5}$$

Vector $\mathbf{x}$ in (5) is an eigenvector of adjacency matrix $\mathbf{A}$, and λ is its corresponding eigenvalue. The convention is to use the eigenvector corresponding to the dominant eigenvalue of $\mathbf{A}$. When the network is directed, the general concern is obtaining a centrality measure based on how often a vertex is being pointed to and the importance of neighbors associated with the incoming edges. Thus with a slight modification to (5), eigenvector centrality is redefined as a vector $\mathbf{x}$ that satisfies

$$\lambda \mathbf{x} = \mathbf{A}'\mathbf{x} \tag{6}$$

where $\mathbf{A}'$ is the transposed adjacency matrix. As discussed in detail in Newman (2010, sec. 7.2), there are several shortcomings to the eigenvector centrality, including that a vertex with no incoming edges will always have centrality of zero. Furthermore, vertices with neighbors that all have zero incoming edges will also have zero centrality because the sum in (4) will not have any nonzero terms.

The Katz–Bonacich centrality, a variation of the eigenvector centrality, seeks to address these issues.

3.5 Katz–Bonacich centrality

The additional inclusion of a free parameter (also called a decay factor) and a vector of exogenous factors into (6) avoids the exclusion of vertices with zero incoming edges while allowing connection values to decay over distance; this is attributed to the culmination of works by Katz (1953), Bonacich (1987), and Bonacich and Lloyd (2001). The centrality measure is defined as a solution to the equation

$$\mathbf{x} = \alpha \mathbf{A}'\mathbf{x} + \boldsymbol{\beta}$$

where α is the free parameter and $\boldsymbol{\beta}$ is the vector of exogenous factors that can either vary or remain constant across vertices. For the centrality measure to converge properly, the absolute value of α must be less than the absolute value of the inverse of the dominant eigenvalue of $\mathbf{A}$. A positive α allows vertices with important neighbors to have higher status, whereas a negative α reduces the status.

4 Clustering coefficient

A clustering coefficient is one way of gauging how tightly connected a network is. The general idea is to consider transitive relations; that is, if vertex j is connected to vertex i and i is connected to k, then j is also connected to k.

Global clustering coefficients provide indication of the degree of concentration of the entire network and consist of overall and average clustering coefficients. The overall clustering coefficient is equal to the number of observed transitive relations divided by the number of possible transitive relations in the network. The average clustering coefficient involves applying the definition of an overall clustering coefficient at the vertex level and then averaging across all the vertices.

For an undirected, unweighted adjacency matrix $\mathbf{A}$, the overall clustering coefficient is defined as

$$c^o(\mathbf{A}) = \frac{\sum_{i;j\neq i;k\neq j;k\neq i} A_{ji}A_{ik}A_{jk}}{\sum_{i;j\neq i;k\neq j;k\neq i} A_{ji}A_{ik}} \tag{7}$$

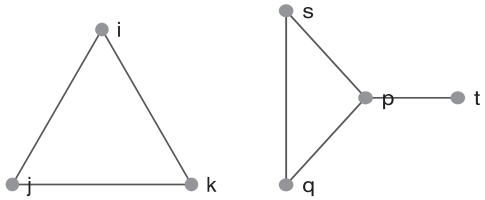
where the numerator represents the sum over i of all closed triplets in which transitivity holds and the denominator represents the sum over i of all possible triplets. With a slight modification in notation, the local clustering coefficient for vertex i is defined as

$$c_i(\mathbf{A}) = \frac{\sum_{j\neq i;k\neq j;k\neq i} A_{ji}A_{ik}A_{jk}}{\sum_{j\neq i;k\neq j;k\neq i} A_{ji}A_{ik}} \tag{8}$$

which leads to the average clustering coefficient:

$$c^a(\mathbf{A}) = \frac{1}{|V|}\sum_i c_i(\mathbf{A}) \tag{9}$$

By convention, $c_i(\mathbf{A}) = 0$ if vertex i has zero links or only one link. Because the average clustering coefficient computes clustering coefficients for each vertex and then takes the average across all vertices, the coefficient gives more weight to low-degree vertices, whereas the overall clustering coefficient takes the average across all triplets.



(a) A closed triplet. $c_i = 1$.

(b) Closed and open triplets. $c_p = 1/3$.

Figure 2. Examples of triplets; drawn using `netplot` (Corten 2011)

Generalized methods exist for computing clustering coefficients. Building upon the works of Barrat et al. (2004), Opsahl and Panzarasa (2009) proposed a set of measures consisting of four types of coefficients, using either the arithmetic mean, geometric mean, maximum, or minimum of the triplet values. Clustering coefficients for vertex i based on the weighted adjacency matrix $\mathbf{W}$ and the corresponding unweighted adjacency matrix $\mathbf{A}$ are calculated as

$$c_i(\mathbf{W}) = \frac{\sum_{j\neq i;k\neq j;k\neq i} \omega A_{jk}}{\sum_{j\neq i;k\neq j;k\neq i} \omega}$$

where the scalar ω equals $(W_{ji}+W_{ik})/2$ for arithmetic mean, $\sqrt{W_{ji} \times W_{ik}}$ for geometric mean, $\max(W_{ji}, W_{ik})$ for maximum, and $\min(W_{ji}, W_{ik})$ for minimum.[3] For unweighted networks, $\mathbf{W} = \mathbf{A}$ and the four types of clustering coefficients are all equal. For unweighted, undirected networks, the overall, local, and average clustering coefficients are equal to (7), (8), and (9), respectively.

5 Maximum flow and minimum cut

The maximum-flow problem, first formulated by Harris and Ross in 1955, involves finding the maximum value that can be routed from the source vertex to the sink vertex, given available paths and capacity constraints along the edges.

In the context of the maximum-flow problem, the adjacency matrix $\mathbf{A}$ is called the capacity matrix. Nonzero A_{ij} entries represent capacity, or the maximum amount of flow that edges can allow through. The flow matrix $\mathbf{F}$ contains entries representing the actual amount of flow that goes through the edges. The residual capacity matrix $\mathbf{R}$ represents the unused edge capacity and is defined as $\mathbf{R} = \mathbf{A} - \mathbf{F}$.

The minimum cut is defined as a partition of graph G into two nonempty sets S and T such that the number of edges (unweighted network) or the aggregation of edge weights (weighted network) connecting S to T is minimal. The value to be minimized is called the weight of the minimum cut, and sets S and T are called sets of source and sink

3. Calculations for the pair $\{W_{ki}, W_{ij}\}$ are also conducted.

vertices, respectively. The minimum-cut set is equal to the set $\{(i,j) \in E | i \in S, j \in T\}$ that minimizes

$$\sum_{(i,j)\in S\times T} A_{ij}$$

The maximum-flow minimum-cut theorem, obtained independently by Elias, Feinstein, and Shannon (1956) and Ford and Fulkerson (1956), states that the minimum-cut weight is equal to the maximum-flow value.

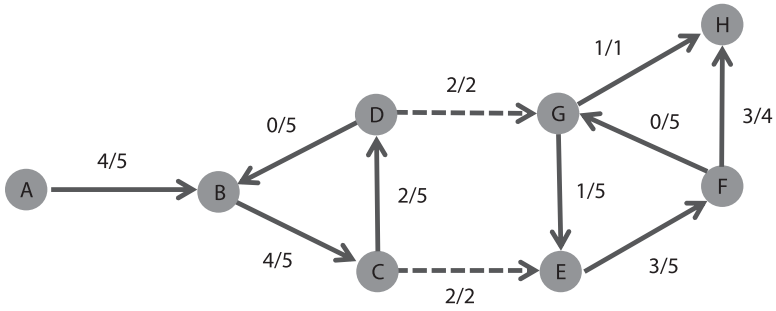


Figure 3. Example network from the Boost graph library (BGL) from Siek, Lee, and Lumsdaine (2001),[4] manually drawn using Microsoft Excel. Fractions represent value of flow in numerator and edge capacity in denominator. A maximum-flow value of 4 is pushed from source vertex A to sink vertex H. Dashed edges (D, G) and (C, E) make up the minimum-cut set with the sum of their capacities equal to 4, consistent with the maximum-flow minimum-cut theorem.

4. http://www.boost.org/doc/libs/1_47_0/libs/graph/doc/graph_theory_review.html#fig:maximum-flow.

6 Mata implementation

6.1 Syntax

struct `matrix_struct` *scalar*	`bfs_sp(`*E*`, `*nodots*`)`
struct `matrix_struct` *scalar*	`dijkstra_sp(`*E*`, `*nodots*`)`
struct `matrix_struct` *scalar*	`floyd_warshall_sp(`*E*`, `*nodots*`)`
real vector	`bfs_betweenness(`*E*`, `*nodots*`)`
real vector	`brandes_dijkstra_betweenness(`*E*`, `*nodots*`)`
struct `matrix_struct` *scalar*	`clustering_coefficients(`*E*`, `*nodots*`)`
struct `matrix_struct` *scalar*	`power(`*E*`, `*nodots*`, `*left*`)`
real vector	`power_katzbonacich(`*E*`, `*alpha*`, `*beta*`, `*nodots*`, `*left*`)`
struct `matrix_struct` *scalar*	`edmonds_karp_max_flow(`*E*`, `*s*`, `*t*`)`
real vector	`bfs_minimum_cut(`*R*`, `*s*`)`
real matrix	`sparse2full(`*E*`, `*initial*`)`
real matrix	`full2sparse(`*X*`)`

where

E:	*real matrix E*	($\|E\| \times 3$ edge list)
alpha:	*real scalar alpha*	(alpha parameter for Katz–Bonacich centrality)
beta:	*real vector beta*	($\|V\| \times 1$ beta vector for Katz–Bonacich centrality)
s:	*real scalar s*	(source vertex for maximum flow)
t:	*real scalar t*	(sink vertex for maximum flow)
R:	*real matrix R*	($\|V\| \times \|V\|$ residual capacity matrix)
X:	*real matrix X*	($\|V\| \times \|V\|$ input matrix)
nodots:	*string scalar nodots*	(optional indicator to suppress display of status dots)
left:	*string scalar left*	(optional indicator to transpose adjacency matrix)
initial:	*real scalar initial*	(optional initial real value of full matrix)

and `matrix_struct` is a structure with real matrix variables `X1` through `X6`.

6.2 Description

This section assumes familiarity with Mata and matrix programming (see [M-0] **intro**).

Structure `matrix_struct` is already compiled as part of the Stata graph library (SGL). Thus users will want to define a variable of type `struct matrix_struct scalar` in their functions to retrieve results from compiled functions that return objects as members of the structure. For example, here is a function that returns a distance matrix for an unweighted network that takes in an edge list matrix with optional parameters for suppressing status dots and replacing missing values:

(Lines enclosed with "`/* */`" denote comments)

```
real matrix distance_matrix(real matrix E, |string scalar nodots,
   real scalar infinity){
   /* Declare variables.  */
   struct matrix_struct scalar mystruct
   real matrix D
   /* Implement breadth-first search algorithm.  */
   mystruct=bfs_sp(E,nodots)
   /* Replace missing values.  */
   D=editmissing(mystruct.X1,infinity)
   /* Return distance matrix.  */
   return(D)
}
```

In the following descriptions, we assume that the user has declared a variable `mystruct` of type `struct matrix_struct scalar`. See [M-2] **struct**.

`bfs_sp(`*E*`, ...)` returns the distance matrix, path matrix, leaf vertex matrix, and adjacency list for an unweighted network in `mystruct.X1`, `mystruct.X2`, `mystruct.X3`, and `mystruct.X4`, respectively. Dimensions of the distance, path, and leaf vertex matrices are $|V| \times |V|$, and dimensions of the adjacency list are $|V| \times m$ where m denotes the largest number of outgoing edges among all vertices in the network.

`dijkstra_sp(`*E*`, ...)` returns the distance and path matrices for a weighted network in `mystruct.X1` and `mystruct.X2`, respectively. Dimensions of the distance and path matrices are $|V| \times |V|$.

`floyd_warshall_sp(`*E*`, ...)` returns the distance matrix for both unweighted and weighted networks in `mystruct.X1`. Dimensions of the distance matrix are $|V| \times |V|$.

`bfs_betweenness(`*E*`, ...)` returns the betweenness centrality vector for an unweighted network. Dimensions of the returned vector are $|V| \times 1$.

`brandes_dijkstra_betweenness(`*E*`, ...)` returns the betweenness centrality vector for a weighted network. Dimensions of the returned vector are $|V| \times 1$.

`clustering_coefficients(`*E*`, ...)` returns the overall and average clustering coefficients in `mystruct.X1` and the local clustering coefficients in `mystruct.X2`. `mystruct.X1` contains a 2×4 matrix, and `mystruct.X2` contains a $|V| \times 4$ matrix.

`power(`*E*`, ...)` returns the dominant eigenvector in `mystruct.X1` and the dominant eigenvalue in `mystruct.X2` if convergence is achieved; the returned eigenvector has dimensions $|V| \times 1$. If convergence is not achieved, Mata error code 3360 is returned in `mystruct.X1`.

`power_katzbonacich(`*E*`, `*alpha*`, `*beta*`, ...)` returns the Katz–Bonacich centrality vector if convergence is achieved and Mata error code 3360 if convergence is not achieved. The scalar *alpha* is the free parameter, and *beta* is the $|V| \times 1$ vector of exogenous factors. The returned Katz–Bonacich centrality vector has dimensions $|V| \times 1$.

`edmonds_karp_max_flow(`E`,` s`,` t`)` returns the maximum-flow value in `mystruct.X1`, $|V| \times |V|$ flow matrix in `mystruct.X2`, and $|V| \times |V|$ residual capacity matrix in `mystruct.X3`. The scalars s and t are the source vertex and the sink vertex, respectively.

`bfs_minimum_cut(`R`,` s`)` returns an indicator vector for the minimum cut with 1 indicating vertices that belong to the set of source vertices and -1 indicating vertices that belong to the set of sink vertices. The returned vector has dimensions $|V| \times 1$.

`sparse2full(`E`, ...)` returns the full matrix corresponding to the inputted edge list matrix $\mathbf{E}$. The returned full matrix has dimensions $|V| \times |V|$.

`full2sparse(`X`)` returns the $|E| \times 3$ edge list matrix corresponding to the full matrix $\mathbf{X}$.

nodots specifies whether to display status dots. If *nodots* is set to a nonempty string, status dots are suppressed. Status dots are displayed if *nodots* is set to "" or is not set.

left specifies that the power method be implemented to calculate the left dominant eigenvector. The right dominant eigenvector is calculated if *left* is set to "" or is not set.

initial specifies initial values to fill the full matrix. If *initial* is not set, the default is to fill the full matrix with all missing values.

6.3 Remarks

The SGL is compiled using Stata version 11.1.

Without loss in generality, all routines assume a directed one-mode network. An undirected edge can be thought of as two directed edges, one going from vertex i to j and another from j to i. Therefore, when working with an undirected network, the edge list matrix $\mathbf{E}$ should have reciprocal relations defined. If reciprocal relations are not defined, such that the edge from vertex i to j is stored but not the edge from j to i, the edge list matrix must be redefined as

$$\mathbf{E} = \texttt{uniqrows}((((\mathbf{E}[.,1] \backslash \mathbf{E}[.,2]), (\mathbf{E}[.,2] \backslash \mathbf{E}[.,1])), (\mathbf{E}[.,3] \backslash \mathbf{E}[.,3]))) \tag{10}$$

where `uniqrows()` returns sorted, unique values. See [M-5] **uniqrows( )**.

Reciprocal relations for weighted networks must be defined with caution. If the initial edge list matrix contains (i, j, w_1) and (j, i, w_2) where $w_1 \neq w_2$, then (10) would produce (i, j, w_1), (j, i, w_2), (j, i, w_1), and (i, j, w_2), resulting in parallel edges and violating one of the assumptions laid out in section 1.2. In such cases, an appropriate adjustment of edge weights may need to be conducted beforehand to avoid generating parallel edges.

The third column consisting of edge weights is required. When connections are intended to represent proximity or intensity, a transformation of those weights may be necessary. If `dijkstra_sp()` is called for an undirected network with edge weights stored as `double` fails to produce symmetric matrices, try converting edge weights to `float`. See [M-5] **floatround( )**.

`bfs_sp()` implements the breadth-first search single-source shortest-path algorithm (the algorithm follows Newman [2010, sec. 10.3]).

`bfs_betweenness()` calls `bfs_sp()` and uses the resulting matrices to compute betweenness centrality (the algorithm follows Newman [2010, sec. 10.3]).

`dijkstra_sp()` implements Dijkstra's single-source shortest-path algorithm based on pseudocode provided by Siek, Lee, and Lumsdaine (2001).[5]

`floyd_warshall_sp()` implements the Floyd–Warshall all-pairs shortest-path algorithm based on pseudocode available from Wikipedia.[6]

`brandes_dijkstra_betweenness()` implements Brandes's (2001) method of calculating betweenness centrality for weighted networks while using Dijkstra's algorithm to find shortest paths.

`clustering_coefficients()` returns a 2×4 matrix in `mystruct.X1` with the first row corresponding to overall coefficients and the second row corresponding to average coefficients. Columns one to four correspond to coefficients calculated using the arithmetic mean, geometric mean, maximum, and minimum, respectively. The $|V| \times 4$ matrix returned in `mystruct.X2` maintains the same column ordering, but the rows correspond to local clustering coefficients instead. When the network is unweighted, the four calculation methods produce the same number.

`power()` and `power_katzbonacich()` routines implement the power method using the sparse matrix or the coordinate list data structure of edge list matrix **E**, respectively. Convergence criteria consist of two rules: There is a maximum of 16,000 iterations, and the maximum relative difference signals convergence if `mreldif()` drops below 1e−10 (see [M-5] **reldif( )**). Depending on the size and density of the network, the power method may provide faster and equally accurate results compared with using Mata's built-in linear algebra functions.

`edmonds_karp_max_flow()` implements the Edmonds–Karp algorithm based on pseudocode available from Wikipedia.[7]

`bfs_minimum_cut()` implements a breadth-first search of the residual capacity matrix returned from running `edmonds_karp_max_flow()` to determine the sets of source and sink vertices.

5. http://www.boost.org/doc/libs/1_47_0/libs/graph/doc/dijkstra_shortest_paths.html.
6. http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm.
7. http://en.wikipedia.org/wiki/Edmonds-Karp_algorithm.

7 Stata implementation (1): SGL wrapper

7.1 Syntax

`netsis` *varname_source varname_target* [*if*] [*in*], `measure(`*network_measure*`)` [`name(`*mata_mname* [, `replace`]`)` `weight(`*weightvar*`)` `label(`*var_prefix* [, `replace`]`)` `directed` `nodots` `infinity(`*real*`)` `power` `notranspose` `alpha(`*real*`)` `beta(`*beta_varlist*`)` `source(`*string*`)` `sink(`*string*`)` `mincut` `residual`]

where *network_measure* can be one of the following:

`adjacency`	adjacency matrix
`distance`	distance matrix
`path`	path matrix
`betweenness`	betweenness centrality
`clustering`	local and overall/average clustering coefficients
`eigenvector`	eigenvector centrality
`maxalpha`	maximum free parameter alpha
`katzbonacich`	Katz–Bonacich centrality
`maxflow`	maximum-flow minimum-cut

varname_source and *varname_target* must be either both numeric or both string type variables. When working with an undirected network, the notions of *source* and *target* do not matter. Furthermore, reciprocal relations for undirected networks do not need to be defined when using the `netsis` command in Stata. Caution must still be exercised when working with weights, however, as outlined in section 6.3. *weightvar* and *beta_varlist* must be type numeric with variables of *beta_varlist* corresponding to the order of *varname_source* and *varname_target*.

`netsis` will return an error code of 198 and exit if parallel edges, self-loops, or nonpositive weights are encountered. Otherwise, the analysis sample will automatically exclude isolated vertices, edges with missing weights (if *weightvar* is specified), and vertices with missing beta exogenous factors (if *beta_varlist* is specified).

7.2 Options

`measure(`*network_measure*`)` specifies the network measure to be computed. `measure()` is required.

`name(`*mata_mname* [, `replace`]`)` specifies the name of the Mata matrix in which to store results. `replace` requests that the existing Mata matrix be overwritten.

`weight(`*weightvar*`)` specifies a numeric edge weight variable. By default, edge weights are set to one for all vertices internally to assume an unweighted network.

`label(`*var_prefix* [, `replace`]`)` specifies that vertex labels be returned to Stata with variable names prefixed with *var_prefix* and suffixed with `_source` and `_target` for

varname_source and *varname_target*, respectively. `replace` also specifies that existing same-named label variables be overwritten.

`directed` specifies directed edges. By default, edges are assumed to be undirected.

`nodots` specifies that status dots be suppressed.

`infinity(`*real*`)` specifies a real number to replace missing or "infinite" distances. Scalars returned in `e()` refer to the matrix before replacement. `name(`*mata_mname*`)` contains the distance matrix after replacement. `infinity()` applies only when generating a distance matrix.

`power` specifies that the power method be implemented in computing the eigenvector centrality, maximum alpha, and Katz–Bonacich centrality. By default, Mata's built-in linear algebra functions are used.

`notranspose` specifies that the adjacency matrix not be transposed when computing the eigenvector and Katz–Bonacich centralities. By default, the adjacency matrix is transposed.

`alpha(`*real*`)` specifies a real number for the free parameter in the Katz–Bonacich centrality calculation. The default is `alpha(1)`.

`beta(`*beta_varlist*`)` specifies exogenous factors for the Katz–Bonacich centrality measure. By default, exogenous factors are set to one for all vertices.

`source(`*string*`)` specifies the source vertex for maximum-flow minimum-cut. The vertex name should be enclosed in double quotes even if it is numeric.

`sink(`*string*`)` specifies the sink vertex for maximum-flow minimum-cut. The vertex name should be enclosed in double quotes even if it is numeric.

`mincut` specifies that the sets of source and sink vertices be determined from the breadth-first search of the residual capacity matrix produced by maximum-flow calculation. An indicator vector is returned instead of the default flow matrix. `mincut` cannot be specified with `residual`.

`residual` specifies that the residual capacity matrix be returned instead of the default flow matrix. `residual` cannot be specified with `mincut`.

7.3 Remarks

`netsis` is a wrapper for class `network_measure`, which is itself a wrapper for functions compiled in SGL. Because SGL functions are each designed to complete specific tasks, a class wrapper is useful in providing additional organization. See [M-2] **class**.

`netsis` does not store results as Stata matrices. See [M-5] **st_matrix( )** for how to transfer Mata matrices from and to Stata.

`netsis` is designed for a one-mode network, which is defined as a network with vertices all of the same type. Projections can be made from a two-mode network (a

network where vertices are classified into two different types) onto a one-mode network by separately considering a network for each type. For example, a two-mode film affiliation network with $|V|_a$ actors and $|V|_m$ movies as vertices can be projected onto two one-mode networks: A $|V|_a$-vertex network with vertices representing actors and edges indicating the appearance of actors in the same film, and a $|V|_m$-vertex network with vertices representing movies and edges indicating movies sharing a common actor (Newman 2010, sec. 6.6). `joinby` can be used to accomplish the projection as shown in section 9.1.

The recommended approach to using the Katz–Bonacich function is to first obtain maximum alpha by using `maxalpha()`, and then based on the returned value, specify the `alpha()` option while calling `katzbonacich()`. Note that singular or near-singular matrices pose computational problems, and thus `eigenvector()`, `maxalpha()`, and `katzbonacich()` functions may not converge in such cases.

..., `measure(clustering) name(`*mata_mname*`)` stores the *mata_mname* matrix in the Mata programming language, where columns one through four correspond to local clustering coefficients based on the arithmetic mean, geometric mean, maximum, and minimum, respectively.

When implementing maximum-flow minimum-cut, the default return object when both the `mincut` and the `residual` option is suppressed is the flow matrix. The `mincut` and `residual` options cannot be specified at the same time.

7.4 Saved results

`netsis, measure(`*network_measure*`)`, where *network_measure* is equal to one of `adjacency`, `distance`, `path`, `betweenness`, or `katzbonacich`, saves the following in `e()`:

Scalars

`e(mean)`	`mean(mean(X)')`	`e(sum)`	`sum(X)`
`e(min)`	`min(X)`	`e(missing)`	`missing(X)`
`e(max)`	`max(X)`	`e(nonmissing)`	`nonmissing(X)`

`netsis, measure(clustering)` saves the following in `e()`:

Matrices

`e(cc)`	overall and average coefficients

`netsis, measure(eigenvector)` saves the following in `e()`:

Scalars

`e(l)`	dominant eigenvalue	`e(sum)`	`sum(X)`
`e(mean)`	`mean(mean(X)')`	`e(missing)`	`missing(X)`
`e(min)`	`min(X)`	`e(nonmissing)`	`nonmissing(X)`
`e(max)`	`max(X)`		

`netsis, measure(maxalpha)` saves the following in `e()`:

Scalars

`e(alpha_max)`	maximum alpha

`netsis, measure(maxflow)` saves the following in `e()`:

Scalars

`e(flow_max)`	maximum-flow value	`e(sum)`	`sum(X)`
`e(mean)`	`mean(mean(X)')`	`e(missing)`	`missing(X)`
`e(min)`	`min(X)`	`e(nonmissing)`	`nonmissing(X)`
`e(max)`	`max(X)`		

In addition, all network measures return the following in `e()`:

Scalars

`e(vertices)`	number of vertices	`e(edges)`	number of edges

Macros

`e(cmd)`	`netsis`	`e(measure)`	network measure
`e(cmdline)`	command as typed	`e(edge)`	`unweighted` or `weighted`
`e(source)`	source vertex variable	`e(network)`	`undirected` or `directed`
`e(target)`	target vertex variable		

Functions

`e(sample)`	marks estimation sample

8 Stata implementation (2): Postcomputation command

8.1 Syntax

`netsummarize` *mata_exp*`, generate(`*newvar_prefix*`) statistic(`*stat_name*`)`

where *stat_name* can be one of the following:

`mean`	`mean(mean(`*mata_exp*`)')`
`min`	`min(`*mata_exp*`)`
`max`	`max(`*mata_exp*`)`
`sum`	`sum(`*mata_exp*`)`
`missing`	`missing(`*mata_exp*`)`
`nonmissing`	`nonmissing(`*mata_exp*`)`
`rowmean`	`mean(`*mata_exp*`')'`
`rowmin`	`rowmin(`*mata_exp*`)`
`rowmax`	`rowmax(`*mata_exp*`)`
`rowsum`	`rowsum(`*mata_exp*`)`
`rowmissing`	`rowmissing(`*mata_exp*`)`
`rownonmissing`	`rownonmissing(`*mata_exp*`)`
`colmean`	`mean(`*mata_exp*`)'`
`colmin`	`colmin(`*mata_exp*`)'`
`colmax`	`colmax(`*mata_exp*`)'`
`colsum`	`colsum(`*mata_exp*`)'`
`colmissing`	`colmissing(`*mata_exp*`)'`
`colnonmissing`	`colnonmissing(`*mata_exp*`)'`

mata_exp must be a Mata matrix or a Mata expression, and it must evaluate to either a scalar, a $|V|\times 1$ column vector, or a $|V|\times|V|$ matrix. New variables *newvar_prefix*`_source`

and *newvar_prefix*`_target` are generated for *varname_source* and *varname_target*, respectively. See [M-5] **sum( )**, [M-5] **mean( )**, [M-5] **missing( )**, and [M-5] **minmax( )**.

8.2 Remarks

Constructing network measures can be time intensive. The postcomputation command allows the user to generate multiple statistics after a network object has been created. For example, after generating a distance matrix **D**, the user can generate variables for closeness centrality with the command[8]

```
netsummarize (rows(D)-1):/rowsum(D), generate(closeness) statistic(rowsum)
```

Specifying `rowmin` or `rowmax` would have also worked because the expression evaluates to a column vector. You also could have set the closeness centrality to a Mata vector beforehand and inserted it as *mata_exp*.

9 Examples

9.1 Creating a one-mode projection from a two-mode network

Here let's illustrate the method of creating a one-mode network out of a two-mode network dataset by using the `joinby` command (see [D] **joinby**). The sample dataset of movies and actors, from the BGL's installation package (Siek, Lee, and Lumsdaine 2001), is based on the *Six Degrees of Kevin Bacon* game created by Fass, Turtle, and Ginelli (1996). `kevin-bacon.dta` contains data for a two-mode network of movies and actors, with edges indicating an actor's appearance in a film. By conducting a projection onto the actors, we can create a one-mode network with vertices representing actors and edges representing the appearance of actors in the same movie. See the result in figure 4.

```
. // Load Kevin Bacon two-mode network dataset.
. use kevin-bacon
(Kevin Bacon two-mode network dataset (Siek et al. 2001))
. // Generate one-mode network of actors.  An edge between actors
. // represents actors appearing in the same movie.
. rename actor actor1
. preserve
. rename actor1 actor2
. save temp_using, replace
(note: file temp_using.dta not found)
file temp_using.dta saved
. restore
. joinby movie using temp_using
. // Remove self-loops.
. drop if actor1==actor2
(91 observations deleted)
```

8. When working with networks involving disconnected components, users may want to specify `(rownonmissing(D)-J(rows(D),1,1)):/rowsum(D)` as the *mata_exp*. Refer to section 3.2.

```
. // Describe data.
. describe
Contains data
  obs:           124                          Kevin Bacon two-mode network
                                                dataset (Siek et al. 2001)
 vars:             3
 size:         8,308                          (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
movie           str33  %33s                   Movie
actor1          str17  %17s                   Actor
actor2          str17  %17s                   Actor
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved
```

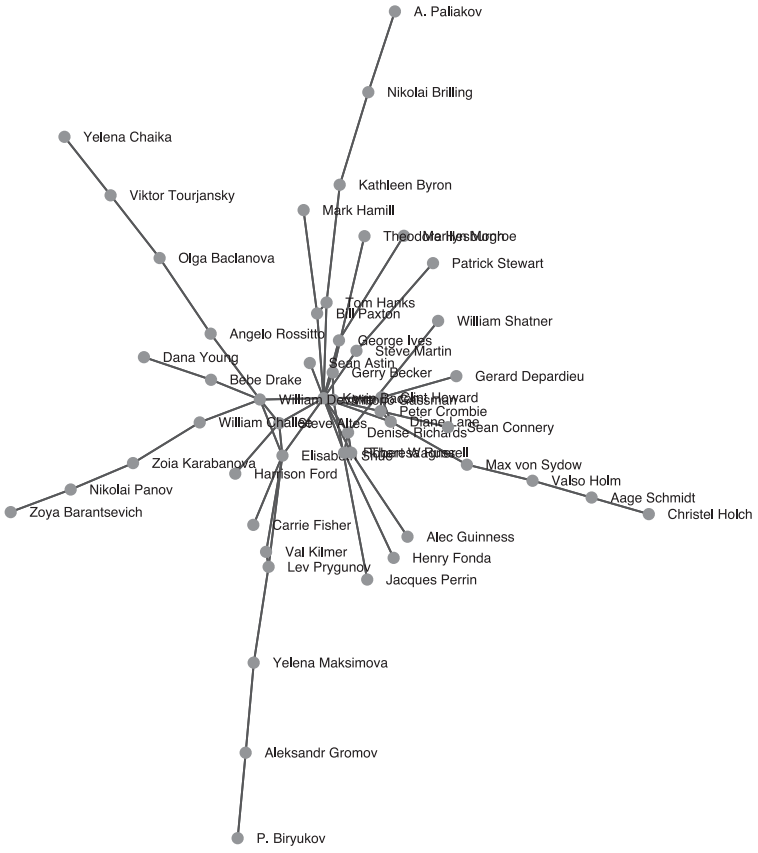


Figure 4. A one-mode network of actors. Sample dataset from BGL (Siek, Lee, and Lumsdaine 2001); drawn using `netplot` (Corten 2011)

9.2 Workflow using pajek2stata, netsis, and netsummarize

Next we create an example workflow using the `pajek2stata` (Corten 2010), `netsis`, and `netsummarize` commands to convert a Pajek (Batagelj and Mrvar 2011) `.net` file to Stata format, and then computing the network centrality measures. See figure 5. The dataset, consisting of fifteenth-century Florentine marriages, is from Padgett and Ansell (1993). The network is undirected and unweighted with the isolated vertex Pucci removed.

```
. // Transfer Pajek´s .net file into Stata using Rense Corten´s
. // pajek2stata.ado.
. pajek2stata using florentine_marriages.net, name(X) clear replace
```

```
Vertices:                          15
Network matrix format:             Edges
Network matrix shape (r x c):      20 X 2
```

```
. // Display edge list saved as Mata matrix.
. mata X
```

	1	2
1	1	2
2	1	3
3	1	4
4	2	3
5	2	5
6	3	6
7	3	4
8	4	7
9	5	8
10	6	8
11	6	9
12	8	9
13	8	10
14	8	11
15	8	12
16	9	7
17	7	13
18	7	10
19	10	14
20	11	15

```
. // Display vertex label.
. list
```

	var1	var2
1.	1	Peruzzi
2.	2	Castellan
3.	3	Strozzi
4.	4	Bischeri
5.	5	Barbadori
6.	6	Ridolfi
7.	7	Guadagni
8.	8	Medici
9.	9	Tornabuon
10.	10	Albizzi
11.	11	Salviati
12.	12	Acciaiuol
13.	13	Lambertes
14.	14	Ginori
15.	15	Pazzi

```
. destring var1, replace
var1 has all characters numeric; replaced as byte
. // Generate new value-label.  See [M-5] st_vlexists().
. mata st_vlmodify("vertex_label",st_data(.,"var1"),st_sdata(.,"var2"))
. label list vertex_label
vertex_label:
           1 Peruzzi
           2 Castellan
           3 Strozzi
           4 Bischeri
           5 Barbadori
           6 Ridolfi
           7 Guadagni
           8 Medici
           9 Tornabuon
          10 Albizzi
          11 Salviati
          12 Acciaiuol
          13 Lambertes
          14 Ginori
          15 Pazzi
. // Store edge list in Stata.
. drop var*
. getmata (source target)=X
. // Assign value label to variables.
. label values source target vertex_label
```

```
. list
```

	source	target
1.	Peruzzi	Castellan
2.	Peruzzi	Strozzi
3.	Peruzzi	Bischeri
4.	Castellan	Strozzi
5.	Castellan	Barbadori
6.	Strozzi	Ridolfi
7.	Strozzi	Bischeri
8.	Bischeri	Guadagni
9.	Barbadori	Medici
10.	Ridolfi	Medici
11.	Ridolfi	Tornabuon
12.	Medici	Tornabuon
13.	Medici	Albizzi
14.	Medici	Salviati
15.	Medici	Acciaiuol
16.	Tornabuon	Guadagni
17.	Guadagni	Lambertes
18.	Guadagni	Albizzi
19.	Albizzi	Ginori
20.	Salviati	Pazzi

```
. // Generate degree centrality.
. netsis source target, measure(adjacency) name(A,replace)
matrix A saved in Mata
. netsummarize A/(rows(A)-1), generate(degree) statistic(rowsum)
. // Generate closeness centrality.
. netsis source target, measure(distance) name(D,replace)
Breadth-first search algorithm (15 vertices)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
...............
Breadth-first search algorithm completed
matrix D saved in Mata
. netsummarize (rows(D)-1):/rowsum(D), generate(closeness) statistic(rowsum)
. // Generate betweenness centrality.
. netsis source target, measure(betweenness) name(b,replace)
Breadth-first search algorithm (15 vertices)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
...............
Breadth-first search algorithm completed
Betweenness centrality calculation (15 vertices)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
...............
Betweenness centrality calculation completed
matrix b saved in Mata
. netsummarize b/((rows(b)-1)*(rows(b)-2)), generate(betweenness)
> statistic(rowsum)
```

```
. // Generate eigenvector centrality.
. netsis source target, measure(eigenvector) name(e,replace)
matrix e saved in Mata
. netsummarize e, generate(eigenvector) statistic(rowsum)
. // Describe and list results.
. describe, full
Contains data
  obs:            20
 vars:            10
 size:           680
```

variable name	storage type	display format	value label	variable label
source	byte	%10.0g	vertex_label	
target	byte	%10.0g	vertex_label	
degree_source	float	%9.0g		rowsum of Mata matrix A/(rows(A)-1)
degree_target	float	%9.0g		rowsum of Mata matrix A/(rows(A)-1)
closeness_source	float	%9.0g		rowsum of Mata matrix (rows(D)-1):/rowsum(D)
closeness_target	float	%9.0g		rowsum of Mata matrix (rows(D)-1):/rowsum(D)
betweenness_source	float	%9.0g		rowsum of Mata matrix b/((rows(b)-1)*(rows(b)-2))
betweenness_target	float	%9.0g		rowsum of Mata matrix b/((rows(b)-1)*(rows(b)-2))
eigenvector_source	float	%9.0g		rowsum of Mata matrix e
eigenvector_target	float	%9.0g		rowsum of Mata matrix e

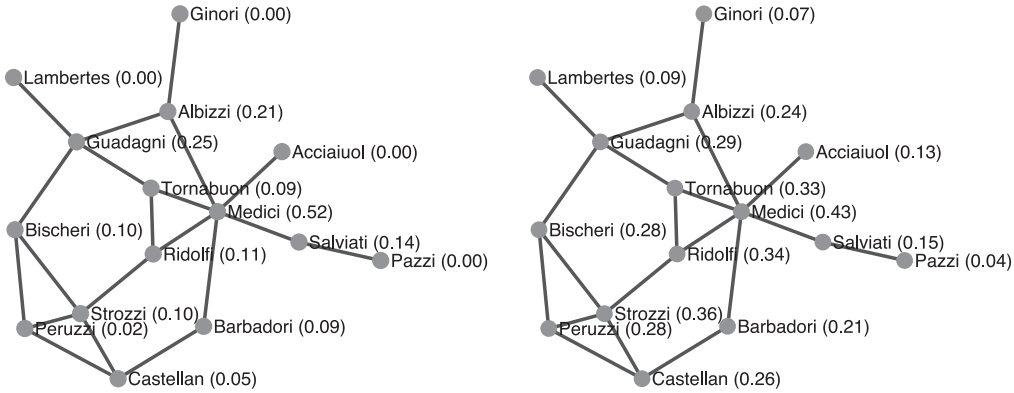
```
Sorted by:
     Note:  dataset has changed since last saved
```

```
. list source target degree* closeness*
```

	source	target	degree~e	degree~t	closen~e	closen~t
1.	Peruzzi	Castellan	.2142857	.2142857	.368421	.3888889
2.	Peruzzi	Strozzi	.2142857	.2857143	.368421	.4375
3.	Peruzzi	Bischeri	.2142857	.2142857	.368421	.4
4.	Castellan	Strozzi	.2142857	.2857143	.3888889	.4375
5.	Castellan	Barbadori	.2142857	.1428571	.3888889	.4375
6.	Strozzi	Ridolfi	.2857143	.2142857	.4375	.5
7.	Strozzi	Bischeri	.2857143	.2142857	.4375	.4
8.	Bischeri	Guadagni	.2142857	.2857143	.4	.4666667
9.	Barbadori	Medici	.1428571	.4285714	.4375	.56
10.	Ridolfi	Medici	.2142857	.4285714	.5	.56
11.	Ridolfi	Tornabuon	.2142857	.2142857	.5	.4827586
12.	Medici	Tornabuon	.4285714	.2142857	.56	.4827586
13.	Medici	Albizzi	.4285714	.2142857	.56	.4827586
14.	Medici	Salviati	.4285714	.1428571	.56	.3888889
15.	Medici	Acciaiuol	.4285714	.0714286	.56	.368421
16.	Tornabuon	Guadagni	.2142857	.2857143	.4827586	.4666667
17.	Guadagni	Lambertes	.2857143	.0714286	.4666667	.3255814
18.	Guadagni	Albizzi	.2857143	.2142857	.4666667	.4827586
19.	Albizzi	Ginori	.2142857	.0714286	.4827586	.3333333
20.	Salviati	Pazzi	.1428571	.0714286	.3888889	.2857143

```
. list source target betweenness* eigenvector*
```

	source	target	betwee~e	betwee~t	eigenv~e	eigenv~t
1.	Peruzzi	Castellan	.021978	.0549451	.2757304	.2590262
2.	Peruzzi	Strozzi	.021978	.1025641	.2757304	.3559805
3.	Peruzzi	Bischeri	.021978	.1043956	.2757304	.2828001
4.	Castellan	Strozzi	.0549451	.1025641	.2590262	.3559805
5.	Castellan	Barbadori	.0549451	.0934066	.2590262	.2117053
6.	Strozzi	Ridolfi	.1025641	.1135531	.3559805	.3415526
7.	Strozzi	Bischeri	.1025641	.1043956	.3559805	.2828001
8.	Bischeri	Guadagni	.1043956	.2545788	.2828001	.2891156
9.	Barbadori	Medici	.0934066	.521978	.2117053	.4303081
10.	Ridolfi	Medici	.1135531	.521978	.3415526	.4303081
11.	Ridolfi	Tornabuon	.1135531	.0915751	.3415526	.3258423
12.	Medici	Tornabuon	.521978	.0915751	.4303081	.3258423
13.	Medici	Albizzi	.521978	.2124542	.4303081	.2439561
14.	Medici	Salviati	.521978	.1428571	.4303081	.1459172
15.	Medici	Acciaiuol	.521978	0	.4303081	.1321543
16.	Tornabuon	Guadagni	.0915751	.2545788	.3258423	.2891156
17.	Guadagni	Lambertes	.2545788	0	.2891156	.0887919
18.	Guadagni	Albizzi	.2545788	.2124542	.2891156	.2439561
19.	Albizzi	Ginori	.2124542	0	.2439561	.0749227
20.	Salviati	Pazzi	.1428571	0	.1459172	.0448134



(a) Betweenness centrality.

(b) Eigenvector centrality.

Figure 5. The centrality measures are reported in parentheses. The network of Florentine marriages is based on data from Padgett and Ansell (1993) and is drawn using `netplot` (Corten 2011).

9.3 Time-series analysis

The use of `if` *exp* allows network measures to be easily generated for subsamples of data. Let's consider a country-level relational dataset and generate overall clustering coefficients for each of the reported years. The dataset is from publicly available International Monetary Fund (IMF) Coordinated Portfolio Investment Survey (CPIS) data, Table 8.1—Geographic Breakdown of Portfolio Investment Assets: Equity Securities (2010). Surveys conducted from 2001 through 2009 are available. The data are provided in matrix format with ij entries pertaining to the amount of country i's equity securities held by country j. Only information on positive investment is used and edge weights are taken to be the inverse of investment deflated using the U.S. consumer price index from the World Bank (2011). Edge weights should reflect proximity between countries that have larger cross-border security holdings. The network is directed.

We focus on the overall measure of clustering coefficients, because nonrespondents from the surveys are included in the network as vertices with only incoming edges, and their local clustering coefficient value of zero causes a bias in the average clustering coefficient toward zero; see figure 6.

```
. // Load IMF CPIS dataset.
. use imfcpis, clear
(IMF Coordinated Portfolio Investment Survey: Table 8.1 - Equity Securities)
. describe

Contains data from imfcpis.dta
  obs:        26,160                          IMF Coordinated Portfolio
                                                Investment Survey: Table 8.1 -
                                                Equity Securities
 vars:             6                          7 May 2011 12:54
 size:     3,793,200                          (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
source          str34  %34s                   Holder of equity securities, IMF
                                                CPIS
target          str95  %95s                   Issuer of equity securities, IMF
                                                CPIS
weight          float  %9.0g                  Edge weight - inverse of deflated
                                                investment in USD, author
year            float  %9.0g                  Year
investment      long   %10.0g                 Year-end holdings of equity
                                                securities in millions of
                                                nominal USD, IMF CPIS
uscpi           float  %9.0g                  US Consumer price index (2005 =
                                                100), World Bank WDI
-------------------------------------------------------------------------------
Sorted by:
. summarize
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
      source |          0
      target |          0
      weight |      26160    1.72e-07    3.07e-07   1.49e-12   1.10e-06
        year |      26160    2005.393    2.531253       2001       2009
  investment |      26160    3479.511    20176.56          1     714928
-------------+---------------------------------------------------------
       uscpi |      26160    101.4221    6.906585    90.6678   110.2466
```

```
. // Compute clustering coefficients for each year.
. // Suppress status dots.
. forvalues i=2001/2009{
  2. di ""
  3. di "** Year: `i´ **"
  4. netsis source target if year==`i´, measure(clustering) directed
> weight(weight) nodots
  5. matrix list e(cc)
  6. }

** Year: 2001 **

Clustering coefficients calculation (170 vertices)
Clustering coefficients calculation completed

e(cc)[2,4]
         Arithmetic~n  Geometric_~n       Maximum       Minimum
Overall     .40182592     .35888101     .41012444     .33131425
Average      .2318437          .2094     .23590407     .19882116
```

(*output omitted*)

```
** Year: 2009 **

Clustering coefficients calculation (195 vertices)
Clustering coefficients calculation completed

e(cc)[2,4]
         Arithmetic~n  Geometric_~n       Maximum       Minimum
Overall     .47241108     .42340634     .48064443     .40074538
Average     .22293559      .1971275     .22709754     .18638563
```

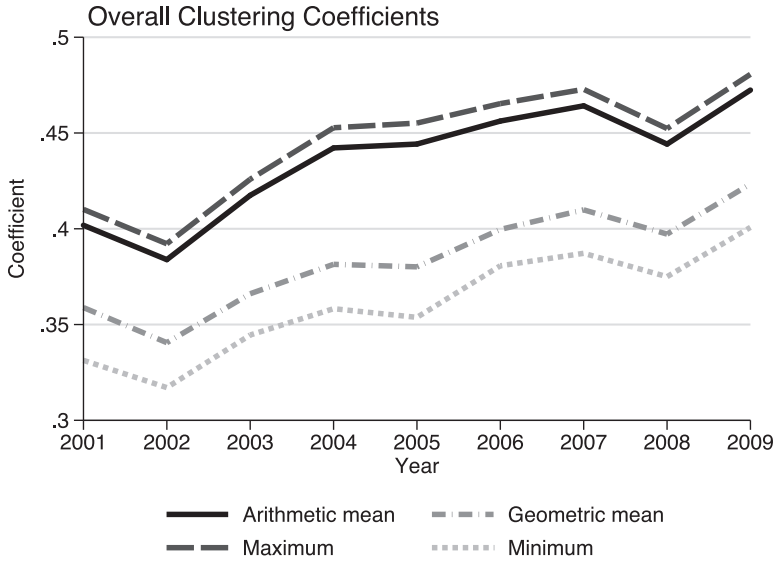


Figure 6. Annual networks of reporting economies from the IMF CPIS data, Table 8.1—Geographic Breakdown of Portfolio Investment Assets: Equity Securities.

9.4 Relational to panel dataset

In this section, we demonstrate an example of a workflow that begins with a relational dataset and results in an unbalanced panel dataset that can be used to run regressions.

The relational data are from the IMF CPIS dataset used in the previous example. Economic, geographic, and governance indicators for the years 2002–2009 are obtained from the World Bank's World Development Indicators and Worldwide Governance Indicators databases (The World Bank 2010, 2011). To facilitate the merging of the two datasets, country names in the World Bank dataset are changed to conform with that of the IMF CPIS dataset. A topic of interest may be to see what factors were important in attracting foreign investment during this period.

Using the IMF CPIS relational dataset for each of the years 2002–2009, we compute ineccentricity and eigenvector centrality. Ineccentricity is defined as the maximum value of incoming shortest paths and is equal to the column maximum of the distance matrix for the weighted, directed network. Because edge weight is equal to the inverse of deflated investment, smaller ineccentricity suggests proximity to neighboring vertices. In computing eigenvector centrality, we use a different type of edge weight instead and use real investment, because eigenvector centrality bestows larger values on more important vertices.

```
. // Load IMF CPIS dataset.
. use imfcpis, clear
(IMF Coordinated Portfolio Investment Survey: Table 8.1 - Equity Securities)
. // Drop year 2001 - World Bank data is available from 2002 to 2009.
. drop if year==2001
(2260 observations deleted)
. // Compute ineccentricity.
. forvalues i=2002/2009{
  2. netsis source target if year==`i´, measure(distance) name(D`i´,replace)
> weight(weight) directed nodots
  3. netsummarize D`i´, generate(ineccentricity`i´) statistic(colmax)
  4. }
Dijkstra´s search algorithm (174 vertices)
Dijkstra´s search algorithm completed
matrix D2002 saved in Mata
  (output omitted)
Dijkstra´s search algorithm (195 vertices)
Dijkstra´s search algorithm completed
matrix D2009 saved in Mata
. // Transfer results to one variable per source and target.
. generate ineccentricity_source=.
(23900 missing values generated)
. label variable ineccentricity_source "ineccentricity"
. generate ineccentricity_target=.
(23900 missing values generated)
. label variable ineccentricity_target "ineccentricity"
```

```
. forvalues i=2002/2009{
  2. quietly replace ineccentricity_source=ineccentricity`i´_source if
> year ==`i´
  3. quietly replace ineccentricity_target=ineccentricity`i´_target if
> year ==`i´
  4. }

. drop *eccentricity200*

. // Compute eigenvector centrality.
. // Generate and use real investment as edge weight.
. generate realinvestment=investment/(uscpi/100)

. forvalues i=2002/2009{
  2. netsis source target if year==`i´, measure(eigenvector)
> name(e`i´,replace) weight(realinvestment) directed nodots
  3. netsummarize e`i´, generate(eigenvector`i´) statistic(rowsum)
  4. }
matrix e2002 saved in Mata
matrix e2003 saved in Mata
matrix e2004 saved in Mata
matrix e2005 saved in Mata
matrix e2006 saved in Mata
matrix e2007 saved in Mata
matrix e2008 saved in Mata
matrix e2009 saved in Mata

. // Transfer results to one variable per source and target.
. generate eigenvector_source=.
(23900 missing values generated)

. label variable eigenvector_source "eigenvector"

. generate eigenvector_target=.
(23900 missing values generated)

. label variable eigenvector_target "eigenvector"

. forvalues i=2002/2009{
  2. quietly replace eigenvector_source=eigenvector`i´_source if year==`i´
  3. quietly replace eigenvector_target=eigenvector`i´_target if year==`i´
  4. }

. drop *eigenvector200*

. rename ineccentricity_target ineccentricity

. rename eigenvector_target eigenvector

. // Collapse by target country and year.
. collapse (sum) weight (sum) investment (sum) realinvestment (mean) uscpi
> (mean) ineccentricity (mean) eigenvector, by(target year)

. // Merge in World Bank data.  Keep matched observations.
. rename target country

. label variable country "Country name"

. merge 1:1 country year using worldbank, keep(matched) nogenerate
    Result                           # of obs.
    -----------------------------------------
    not matched                             0
    matched                             1,287
    -----------------------------------------

. // Declare data to be a panel.
. encode country, generate(country2)
```

```
. xtset country2 year
       panel variable:  country2 (unbalanced)
        time variable:  year, 2002 to 2009, but with gaps
                delta:  1 unit

. label data "Country-level panel data"

. describe

Contains data
  obs:         1,287                          Country-level panel data
 vars:            20
 size:       284,427                          (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
country         str95  %95s                   Country name
year            float  %9.0g                  Year
weight          double %9.0g                  (sum) weight
investment      double %10.0g                 (sum) investment
realinvestment  double %9.0g                  (sum) realinvestment
uscpi           float  %9.0g                  (mean) uscpi
ineccentricity  float  %9.0g                  (mean) ineccentricity
eigenvector     float  %9.0g                  (mean) eigenvector
exrate          float  %9.0g                  DEC alternative conversion factor
                                                (LCU per US$)
gdpgrowth       float  %9.0g                  GDP growth (annual %)
inflation       float  %9.0g                  Inflation, consumer prices
                                                (annual %)
region          str26  %26s                   Region
incomegroup     str20  %20s                   Income Group
corruption      float  %9.0g                  Control of Corruption: Estimate
effectiveness   float  %9.0g                  Government Effectiveness:
                                                Estimate
stability       float  %9.0g                  Political Stability and Absence
                                                of Violence/Terrorism: Estimate
regulation      float  %9.0g                  Regulatory Quality: Estimate
ruleoflaw       float  %9.0g                  Rule of Law: Estimate
accountability  float  %9.0g                  Voice and Accountability:
                                                Estimate
country2        long   %38.0g       country2  Country name
-------------------------------------------------------------------------------
Sorted by:  country2  year
     Note:  dataset has changed since last saved
```

```
. summarize
```

Variable	Obs	Mean	Std. Dev.	Min	Max
country	0				
year	1287	2005.625	2.278011	2002	2009
weight	1287	2.95e-06	2.52e-06	4.65e-09	.0000121
investment	1287	64439.03	218706.4	1	2383743
realinvest~t	1287	62613.31	210932.3	.9070571	2245200
uscpi	1287	101.9301	6.458229	92.10583	110.2466
ineccentri~y	1287	3.52e-07	2.52e-07	3.71e-08	1.47e-06
eigenvector	1287	.0249071	.0746418	0	.5546467
exrate	1222	506.7404	1860.914	.268788	17065.08
gdpgrowth	1214	4.225663	5.11624	-41.3	40.79159
inflation	1124	29.91726	728.86	-13.22581	24411.03
region	0				
incomegroup	0				
corruption	1260	.1205744	1.031661	-2.016047	2.466556
effectiven~s	1260	.1524394	1.006756	-2.249335	2.267191
stability	1270	.0321373	.9800003	-2.880965	1.596395
regulation	1260	.1565138	.9833775	-2.691515	1.992202
ruleoflaw	1272	.0948148	.9989472	-2.576503	1.964045
accountabi~y	1271	.0841978	.9925295	-2.290506	1.826686
country2	1287	97.04817	54.78466	1	191

9.5 Maximum flow and minimum cut

Here we demonstrate a method of obtaining the flow matrix, residual capacity matrix, and the vector indicating sets of source and sink vertices of the minimum cut of the example network presented in section 5. The directed edge list includes a third column for edge capacity.

```
. // Load max-flow min-cut network dataset.
. use max-flow_min-cut, clear
(Max-flow min-cut network dataset (Siek et al. 2001))
. // Generate and display flow matrix.
. netsis v1 v2, measure(maxflow) weight(capacity) directed source("A")
> sink("H") name(F,replace)
Edmonds-Karp algorithm
Edmonds-Karp algorithm completed
matrix F saved in Mata
. mata F
```

	1	2	3	4	5	6	7	8
1	0	4	0	0	0	0	0	0
2	0	0	4	0	0	0	0	0
3	0	0	0	2	2	0	0	0
4	0	0	0	0	0	0	2	0
5	0	0	0	0	0	3	0	0
6	0	0	0	0	0	0	0	3
7	0	0	0	0	1	0	0	1
8	0	0	0	0	0	0	0	0

```
. // Generate and display residual capacity matrix.
. netsis v1 v2, measure(maxflow) weight(capacity) directed source("A")
> sink("H") name(R,replace) residual

Edmonds-Karp algorithm
Edmonds-Karp algorithm completed
matrix R saved in Mata

. mata R
       1   2   3   4   5   6   7   8
  +-------------------------------------+
1 |   0   1   0   0   0   0   0   0   |
2 |   0   0   1   0   0   0   0   0   |
3 |   0   0   0   3   0   0   0   0   |
4 |   0   5   0   0   0   0   0   0   |
5 |   0   0   0   0   0   2   0   0   |
6 |   0   0   0   0   0   0   5   1   |
7 |   0   0   0   0   4   0   0   0   |
8 |   0   0   0   0   0   0   0   0   |
  +-------------------------------------+

. // Generate variables indicating sets of source and sink vertices.
. netsis v1 v2, measure(maxflow) weight(capacity) directed source("A")
> sink("H") name(m,replace) mincut

Edmonds-Karp algorithm
Edmonds-Karp algorithm completed

Breadth-first search algorithm
Breadth-first search algorithm completed
matrix m saved in Mata

. netsummarize m, generate(mincut) statistic(rowsum)

. list

     +-------------------------------------------------------+
     | v1   v2   capacity   mincut~e   mincut~t |
     |-------------------------------------------------------|
  1. |  A    B          5          1          1 |
  2. |  B    C          5          1          1 |
  3. |  C    D          5          1          1 |
  4. |  C    E          2          1         -1 |
  5. |  D    B          5          1          1 |
     |-------------------------------------------------------|
  6. |  D    G          2          1         -1 |
  7. |  E    F          5         -1         -1 |
  8. |  F    G          5         -1         -1 |
  9. |  F    H          4         -1         -1 |
 10. |  G    E          5         -1         -1 |
     |-------------------------------------------------------|
 11. |  G    H          1         -1         -1 |
     +-------------------------------------------------------+

. // Make sure minimum cut weight equals maximum flow value as should be the
. // case by the max-flow min-cut theorem.
. egen mincap=sum(capacity) if mincut_source==1 & mincut_target==-1
(9 missing values generated)

. quietly sum mincap

. assert r(mean)==e(flow_max)
```

10 Function and performance testing

10.1 Testing results using C++ and R

The following table (table 1) provides information about packages used to check network measures for each type of network. These packages include the BGL in C++ (Siek, Lee, and Lumsdaine 2001) and `sna`/`igraph` in R (Butts 2010; Csardi and Nepusz 2006).[9] For each of the directed and undirected networks, function tests are conducted for weighted and unweighted networks, for a series of random networks with $|V|$ starting from approximately 100 and increasing to a maximum of 500 in increments of about 50 vertices. Maximum density, defined as $|E|/|V|^2$ with undirected edges counted as two directed edges, is set to 0.1 in all networks.

For clustering coefficients, "L" denotes local, "A" denotes average, and "O" denotes overall. "NA"s are inserted for cases where the packages used do not compute network measures for the particular network type or in cases where the network structure is computationally problematic, such as for packages that differ in handling near-singular matrices.

9. We use BGL following Gleich (2008) and Carey, Long, and Gentleman (2011). Programs used for function and performance tests are available at https://sites.google.com/site/statagraphlibrary/sgl-qa.

Table 1. Function tests involving BGL are successful if the network measures generated using SGL and BGL are equal. For tests involving R packages, the success condition is evaluated for when maximum relative difference, `mreldif()`, between SGL and measures produced in R is less than 1e−7. See [M-5] **reldif( )**.

	Unweighted		Weighted	
	Undirected	Directed	Undirected	Directed
Adjacency matrix	BGL	BGL	BGL	BGL
Distance matrix	BGL	BGL	BGL	BGL
Path matrix	BGL	BGL	BGL	BGL
Betweenness centrality	BGL	BGL	BGL	BGL
Eigenvector centrality	`igraph`	NA	`igraph`	NA
Katz–Bonacich centrality	`igraph`	`igraph`	`igraph`	`igraph`
Clustering coefficient	`igraph` (L,A,O)	`sna` (O)	NA	NA
Maximum-flow minimum-cut	BGL	BGL	BGL	BGL

10.2 Performance tests

Time complexity is an issue, especially for weighted networks. In the function tests completed above, the current SGL algorithm implementation times for computing the distance matrix and betweenness centrality increase exponentially when the number of vertices increases. Illustrations below (in figure 7) demonstrate algorithm completion times for both SGL and BGL routines on directed, weighted networks. Similar figures occur for undirected, weighted cases.

The difference between SGL and BGL completion times is not as stark for unweighted networks. Although SGL routines take longer than BGL, the divergence is not as great as what is reported from tests involving weighted networks. For a random directed, unweighted network of approximately 500 vertices, computing the distance matrix and betweenness centrality takes about 10.93 and 26.03 seconds, respectively, whereas the corresponding completion times are about 5.18 and 5.86 seconds using BGL. Similar figures occur for undirected, unweighted networks.

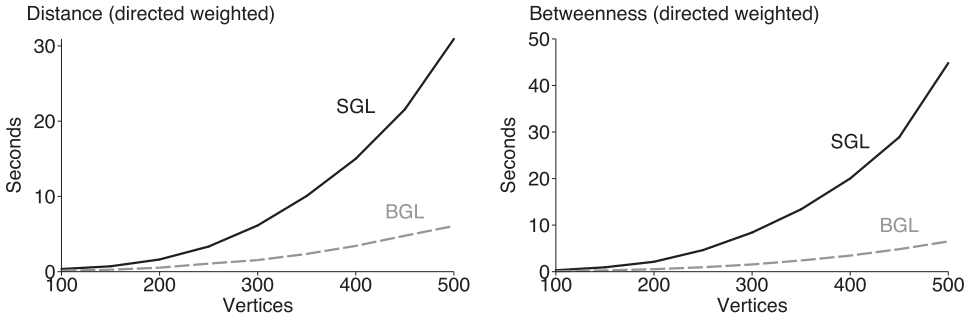


Figure 7. Distance matrix and betweenness centrality computations for randomly generated directed, weighted networks with maximum density of 0.1. Tests are conducted for network sizes in increments of approximately 50 vertices from $|V| \sim 100$ to $|V| \sim 500$. SGL is run on 4-core Stata/MP version 11.1 and uses Dijkstra's single-source shortest-path algorithm for both computations with methodology from Brandes (2001) applied to calculate the betweenness centrality. BGL uses Boost version 1.47.0 and implements Johnson's all-pairs shortest-path algorithm (Johnson 1977) to compute the distance matrix and Brandes' algorithm to calculate betweenness centrality. An adjacency list graph structure is used to represent the network in BGL. Both libraries are compiled and run on a 64-bit Linux operating system.

11 Conclusion and future developments

The SGL demonstrates the ability to use relational data and generate network measures in Stata. As shown in the examples, relational data can be constructed using existing Stata commands such as `joinby` or by using user-written commands such as Rense Corten's `pajek2stata`. Network measures computed by `netsis` and returned to Stata by `netsummarize` can be merged to datasets for running regressions.

Though SGL, `netsis`, and `netsummarize` work in unison to provide usable network information, significant work remains to make the process more efficient and to provide additional enhancements. Areas for future development include implementing more efficient algorithms, designing algorithms for additional network measures, and optimizing SGL in Mata. It is hoped that further improvements and expansions will help facilitate the analysis of networks using Stata.

12 Acknowledgments

Special thanks to Phil Schumm for providing me with guidance on the structure of the program, and Rense Corten for his valuable feedback on the program and the initial draft article. I wish to thank an anonymous reviewer for helping me improve the program and the documentation. I am grateful for Siek, Lee, and Lumsdaine (2001, BGL), Gleich

(2008, MatlabBGL), Butts (2010, `sna`), and Csardi and Nepusz (2006, `igraph`) for making their programs and source codes available. I would like to express my profound gratitude to Galina Hale for introducing me to network analysis and supporting me throughout my work on this project.

13 References

Anthonisse, J. M. 1971. The rush in a directed graph. Technical Report BN 9/71, Stichting Mathematisch Centrum, Amsterdam.

Barrat, A., M. Barthélemy, R. Pastor-Satorras, and A. Vespignani. 2004. The architecture of complex weighted networks. *Proceedings of the National Academy of Sciences* 101: 3747–3752.

Batagelj, V., and A. Mrvar. 2011. Pajek. http://pajek.imfm.si/doku.php?id=download.

Bonacich, P. 1972. Factoring and weighting approaches to status scores and clique identification. *Journal of Mathematical Sociology* 2: 113–120.

———. 1987. Power and centrality: A family of measures. *American Journal of Sociology* 92: 1170–1182.

Bonacich, P., and P. Lloyd. 2001. Eigenvector-like measures of centrality for asymmetric relations. *Social Networks* 23: 191–201.

Brandes, U. 2001. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology* 25: 163–177.

Butts, C. T. 2010. *sna: Tools for Social Network Analysis*. R package version 2.2-0. http://cran.r-project.org/web/packages/sna/index.html.

Carey, V., L. Long, and R. Gentleman. 2011. *RBGL: An interface to the BOOST graph library*. R package version 1.28.0. http://cran.r-project.org/web/packages/RBGL/index.html.

Corten, R. 2010. pajek2stata: Stata module to import network data in Pajek's .net format. Statistical Software Components S457149, Department of Economics, Boston College. http://econpapers.repec.org/software/bocbocode/s457149.htm.

———. 2011. Visualization of social networks in Stata using multidimensional scaling. *Stata Journal* 11: 52–63.

Csardi, G., and T. Nepusz. 2006. The igraph software package for complex network research. InterJournal. http://interjournal.org/manuscript_abstract.php?361100992.

Elias, P., A. Feinstein, and C. E. Shannon. 1956. A note on the maximum flow through a network. *IRE Transactions on Information Theory* 2: 117–119.

Fass, C., B. Turtle, and M. Ginelli. 1996. *Six Degrees of Kevin Bacon*. New York: Plume.

Ford, L. R., Jr., and D. R. Fulkerson. 1956. Maximal flow through a network. *Canadian Journal of Mathematics* 8: 399–404.

Freeman, L. C. 1977. A set of measures of centrality based on betweenness. *Sociometry* 40: 35–41.

Gleich, D. 2008. MatlabBGL. http://www.mathworks.com/matlabcentral/fileexchange/10922.

Harris, T. E., and F. S. Ross. 1955. Fundamentals of a method for evaluating rail net capacities. Research Memorandum No. 1573, The RAND Corporation, Santa Monica, CA.

International Monetary Fund. 2010. Coordinated Portfolio Investment Survey (CPIS). http://cpis.imf.org/.

Johnson, D. B. 1977. Efficient algorithms for shortest paths in sparse networks. *Journal of the Association for Computing Machinery* 24: 1–13.

Katz, L. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18: 39–43.

Newman, M. E. J. 2010. *Networks: An Introduction*. Oxford: Oxford University Press.

Opsahl, T., and P. Panzarasa. 2009. Clustering in weighted networks. *Social Networks* 31: 155–163.

Padgett, J. F., and C. K. Ansell. 1993. Robust action and the rise of the Medici, 1400–1434. *American Journal of Sociology* 98: 1259–1319.

Siek, J., L.-Q. Lee, and A. Lumsdaine. 2001. The Boost Graph Library (BGL). http://www.boost.org/libs/graph/doc/index.html.

The World Bank. 2010. Worldwide Governance Indicators (WGI) Online. http://data.worldbank.org/data-catalog/worldwide-governance-indicators.

———. 2011. World Development Indicators (WDI) Online. http://data.worldbank.org/data-catalog/world-development-indicators.

About the author

Hirotaka Miura is a graduate student in the Department of Mathematics at Columbia University studying for a masters degree in mathematical finance.